Integrated Parallel Macros
User's Guide

Luc Giraud, Pascale Noyret,

Eric Sevault, Vincent Van Kemenade

January 1995 TR/PA/95/01

IPM 2.3
User's Guide
and Reference Manual

January 1995

english:
Integrated **P**arallel **M**acros

french:
Inutile **P**aquet de **M**acros

Abstract

This user's guide describes the features of the set of m4 macros provided by IPM (Integrated Paralell Macros). Those macros allow the users to easily implement a portable distributed programme based on the message passing paradigm using a hidden message passing library like PVM, PARMACS or a subset of MPI. The choice of the hidden message passing library is made only at the compilaton time.
The purpose of this set of macros is not to provide the user with a new message passing library (IPM is not even a library) but only to provide the user with a simple message passing interface. Some IPM macros give an easy access to relevant message passing data, while avoiding a deep technical knowledge of the hidden message passing library.
We made the choice of a set of macros (although the drawback of some possible border effects) instead of a library, mainly because it allows the advanced users to have access to the expanded code for tuning some communication according to his knowledge of both the target architecture and the "hidden" message passing library.

L. Giraud, P. Noyret, E. Sevault, V. Van Kemenade

- CERFACS -

42 avenue Coriolis, 31 057 TOULOUSE cedex

Contents

Acknowledgements

Some important contributions to the definition and the continuing development of IPM have come from researchers who do not formally belong to the "FunParTools Corporation". We would like to thank the following persons for their help and suggestions:

Norman Barth (UCSD)

Frederic Carbonnell (CERFACS)

Tuomo Kauranne (University of Joensuu-Finland)

Jean Latour (CRAY FRANCE)

Michael Rudgyard (CERFACS)

Sami Saarinen (CSC-Finland)

Philip Hariss (FEGS-Cambridge)

1. Introduction

This guide contains a brief overview of the information required to implement parallel code using message passing with IPM 2.3. A dominant characteristic of the "Integrated Parallel Macros" is that they are "integrated" macros, and are therefore easy to use (at least, in our opinion !!). The user may then avoid many of the technical aspects of the parallel implementation and concentrate on the application itself.

IPM is not a new message passing library. Not even a library. It is just a package of m4[1] macros built on top of existing message passing libraries such as PVM[2] 3.2.x or 3.3.x or PARMACS[3] 6.0. It allows the user to choose between parallel libraries at compilation time although he retains a single source code – it therefore reduces development costs while permitting the use of libraries that may be better suited for a particular hardware configuration. New releases of these libraries are hidden from the user because of the compact IPM interface. For advanced features, the user can easily alter parts of the macros or modify the expanded source code before compilation.

IPM has been designed keeping in mind the specifications for MPI. Converting applications from IPM to MPI should not be difficult — indeed, this will not be necessary as soon as an IPM interface to MPI is available. An MPI subset, built on top of IPM calls, is provided with the IPM package.

2. Conventions

2.1. IPM Programming Models

IPM supports both Host-Node and SPMD programming models:

- For a Host-Node scheme, the user can explicitly define the environment through the IPM_BEGIN_HOST or the IPM_BEGIN_NODE calls. All others macros can be invoked from the host or node program.

  The nodes are created using the IPM_CREATE_TORUS or IPM_CREATE_GRAPH macros, which define a logical topology for node processes. In many cases, the host may be excluded from this logical topology and therefore does not participate in the computational part of the execution. Note that the IPM_BARRIER macro provides the user with a barrier that synchronizes all the node processes.

- With the SPMD model, each process runs the same program. Since IPM is built on top of parallel libraries where one process is in charge of the creation of the others, there is always a "host" process, even if this is hidden from the user. As a result, the work to be performed in some macros is only defined at execution time.

[1] m4 is the standard Unix macro language processor.
[2] Geist et al. Oak Ridge National Laboratory.
[3] Hempel et al. Institute fur Methodische Grundlagen and GMD

For example, the IPM_INIT macro performs functions of IPM_BEGIN_HOST or IPM_BEGIN_NODE depending on the nature ("host/node") of the calling process. For the same reason, the IPM_CREATE instructions are ignored when running on a "node" process. The "host" process is supposed to be a formal member of the logical topology and has an effective part of the computational work. The IPM_GBARRIER synchronizes both "host" and "node" processes.

2.2. Process Identifier

All the processes are identified by an integer rank, starting from 0 up to *size*-1, where *size* is the total number of processes. In most cases, one can distinguish host process with rank 0 and node processes with ranks from 1 up to *number_of_nodes*. These logical values are the only ones recognized by IPM. The underlying task identifiers (*tid* PVM) or process identifiers (*pid* PARMACS) are no longer required and are completely hidden from the user.

2.3. Logical Topologies

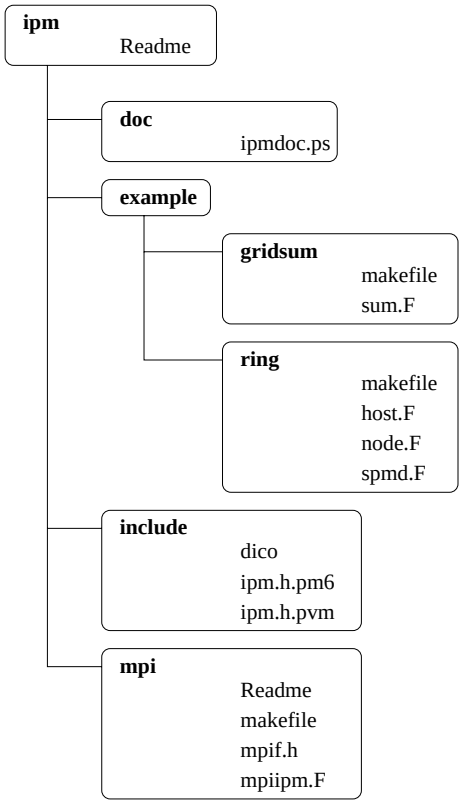
As well as MPI and PARMACS, IPM provides two main topologies: a torus of up to four dimension and a customized graph. As these features are not available with PVM, they are completely emulated by IPM. A binary tree is also provided for global reduction or scatter operations across the nodes.

2.4. Communications

Processes exchange data through tagged messages. IPM provides asynchronous send, and blocking or non-blocking receive functions, as well as multi-cast and probe functions. These communication routines allow the user to communicate contiguous data in the memory: no packing is assumed and the user may need to define intermediate "buffer" arrays before using these functions. A message is defined by its datatype and the number of elements sent or received (previous releases of IPM only needed the length in bytes). Processes ranks are used to specify the destination or the origin of messages.

3. Obtaining IPM

The IPM package is available by anonymous ftp from orion.cerfacs.fr (138.63.200.33) on /pub/algo/softw/ipm. The file is available in a tar compressed (or gzipped) format "ipm.tar.Z" (or ipm.tar.gz). After an *uncompress* and a *tar xvf ipm.tar*, one can retrieve the following directory:



The file *ipm/include/dico* contains the list of all the macros defined in the include files. The *mpi* directory contains a subset of MPI functions built on top of IPM. See the *Readme* file of this directory for more information.

4. Compiling IPM Applications

Before compiling an application written with IPM, the user has to select an available parallel library for the target machine. A symbolic link must then be defined between the local ipm.h file and the corresponding IPM include file. For example, if the library to be used is PVM, the following link should be created :

```
ln -s $HOME/ipm/include/ipm.h.pvm ipm.h
```

The user should also link with the specific include files of the chosen library such as "fpvm3.h" (PVM) or "pm.inc" (PARMACS 6.0). Files with IPM calls must contain one (and only one) m4 *include* instruction at the beginning of the file:

```
include(ipm.h)
c
      subroutine subvoid
c     ...fortran code...
```

These files should be expanded before compilation using the m4 pre-processor in the makefile (an example of a generic makefile using IPM is given in appendix II):

```
.F.o:
        m4 $*.F | egrep -v '#'  > $*.f
        $(F77) $(OPTIONS) -c $*.f
```

5. User Interface

5.1. Naming Conventions

All of the macros start with "IPM_" prefix and are written in upper case letters. No blanks or carriage returns are permitted in the macro interface:

```
c     correct:
      IPM_GET_RANK(coords,rank)
c
c     incorrect:
      IPM_GET_RANK ( coords, rank )
      IPM_GET_RANK(coords,
     *             rank)
```

5.2. Behavior of Macros Calls

Some macros behave like subroutine calls and others as functions calls. For example,

```
IPM_GET_RANK(coords,rank)
```

behaves as a subroutine call, where *rank* is an "output" argument of the subroutine, and

```
neigh = IPM_RANK + 1
```

behaves as a function call. Note that there is no overhead when functions such as IPM_RANK are called several times, and it would be against IPM's philosophy to store this value elsewhere.

5.3. List of Macros

Process management

IPM_ABORT

IPM_BARRIER

IPM_BEGIN_HOST

IPM_BEGIN_NODE

IPM_DECLAR

IPM_END

IPM_INIT
IPM_GBARRIER

Logical topology

IPM_CREATE_GRAPH
IPM_CREATE_TORUS

Communication features

IPM_MCAST
IPM_NRECV
IPM_PROBE
IPM_RECV
IPM_SEND

Global tree

IPM_DADDY
IPM_MAXSONS
IPM_NB_SONS
IPM_ROOT
IPM_SONS

Inquiry functions

IPM_GET_ALL_COORDS
IPM_GET_COORDS
IPM_GET_RANK
IPM_HOST
IPM_HOST_STATUS
IPM_LIBRARY
IPM_MSG_RECV
IPM_MSG_SEND
IPM_NBRECV

IPM_NBSEND
IPM_NB_NEIGHBORS
IPM_NEIGHBORS
IPM_NODES
IPM_RANK
IPM_SIZE
IPM_TOPO
IPM_TORDIM
IPM_TORSIZE
IPM_TYPE_LENGH
IPM_VERSION

Defined Constants

IPM_ANYSND
IPM_ANYTAG
IPM_GRAPH
IPM_HOST_I
IPM_HOST_0
IPM_NO_DADDY
IPM_NONE
IPM_PARMACS
IPM_PVM
IPM_TORUS

Generic Data–types

IPM_COMPLEX
IPM_DBLE
IPM_INTEGER
IPM_REAL

Specific Data–types

IPM_BYTE
IPM_CHARACTER

IPM_COMPLEX8

IPM_COMPLEX16

IPM_INT2

IPM_INT4

IPM_INT8

IPM_REAL4

IPM_REAL8

We refer to the appendix for a complete description of those macros.

6. Advanced Features

6.1. Data–types

In order to run on computers with different intrinsic data representation types, one can edit the ipm.h file and modify the generic data types:

```
# for 4-bytes reals (Sun4s, IBM RS6000s, ...): IEEE arithmetic.
define(IPM_REAL,REAL4)
# for 8-bytes reals (Crays)
#define(IPM_REAL,REAL8)
```

The sources of the application itself need not be changed if generic data types are used in communication macros.

6.2. Encoding flag in ipm.h.pvm

The PVM encoding scheme when initializing send buffers is defined in an internal macro of the ipm.h.pvm file, which may be edited and modified:

```
# Initsend Buffer Flag ( Customized )
# -------- ------ ----
#
define(IPM_INITFLAG,PVMRAW)
```

The PVMINPLACE option is not available with PVM 3.2.

6.3. Internal integer type in ipm.h.pvm

IPM uses and exchanges integers to store process identifiers, current topology, trees information...etc. In order to keep a maximum of flexibility, no specific type had been defined for these integers. An integer is then defined by two IPM internal macros: an internal type and the number of internals needed to constitute an integer. The default values are:

```
# Internal Integer Type and Basic unit ( Customized )
# -------- ------- ---- --- ----------
#
define(IPM_INTERNAL,INTEGER4)
define(IPM_INTBUNIT,1)
```

which defines the internal integer used by IPM to be a single INTEGER4. If the user wishes to run on a CRAY T3D, or other machines where 8 bytes integers are default, he should define:

```
define(IPM_INTERNAL,BYTE1)
define(IPM_INTBUNIT,8)
```

7. Warnings and Bugs

A customized graph which includes the host process as an active process is not possible using PARMACS 6.0. The only way of running such an application is to define an underlying one-dimensional torus with PARMACS and then to emulate the customized graph.

If the current mapping is a torus and the host process is not included in this topology, then calling IPM_GET_COORDS with the host process' rank returns the point of origin. However, IPM_GET_RANK with the same coordinates returns 1, which is clearly not the rank of the host process.

It happens on the Meiko CS-2 machine that an executable is unable to spwan itself. No definitive reason has been found for this problem. A solution is to duplicate the executable.

The user should not use message tags greater than 16000000.

Appendix I:
Reference pages for IPM 2.3 macros

Synopsis

IPM_ABORT

Arguments

IN

none

OUT

none

Discussion

The IPM_ABORT macro kills all existing processes except itself and exits from the parallel application. The calling process is not killed.

Example

```
c
c*    Check for incorrect value of IPM_RANK
c
      if ( IPM_RANK.lt.0 ) then
         IPM_ABORT
         stop
      endif
```

Synopsis

IPM_ANYSND

Arguments

IN

none

OUT

none

Discussion

Using IPM_ANYSND in place of the sender's rank in *receive* and *probe* functions matches any process rank.

Example

```
c
c*    Check if a specified tagged message has arrived
c
      integer msgtag, is, it, arrived
c
      msgtag = 101
      IPM_PROBE(IPM_ANYSND,msgtag,is,it,arrived)
```

Synopsis

IPM_ANYTAG

Arguments

IN

none

OUT

none

Discussion

Using IPM_ANYTAG in place of the tag argument in *receive* and *probe* functions matches any tag number.

Example

```
c
c*    Check if any kind of message has arrived
c
      integer is, it, arrived
c
      IPM_PROBE(IPM_ANYSND,IPM_ANYTAG,is,it,arrived)
```

Synopsis

IPM_BARRIER(bartag)

Arguments

IN

bartag - integer barrier identifier supplied by the user. Should be in the same range than message tag. It allows the user to distinguish between two barriers.

OUT

none

Discussion

IPM_BARRIER synchronizes all the node processes. If called from the host program, the barrier call is ignored. Node processes have to use the same barrier identifier, otherwise a deadlock will occur.
See also IPM_GBARRIER.

Example

```
c
c*    first barrier
      IPM_BARRIER(101)
c
c*    second barrier
      IPM_BARRIER(102)
```

Synopsis

IPM_BEGIN_HOST

Arguments

IN

none

OUT

none

Discussion

IPM_BEGIN_HOST is the first macro invoked before any other IPM call in the host program. It performs first initial set-up of basic enquiry functions.
See also IPM_INIT.

Example

```
      program host
c
include(ipm.h)
c
      IPM_DECLAR
c
      IPM_BEGIN_HOST
c
      write(6,*) 'size = ', IPM_SIZE
      write(6,*) 'nodes= ', IPM_NODES
c
      IPM_END
      end
```

Synopsis

IPM_BEGIN_NODE

Arguments

IN

none

OUT

none

Discussion

IPM_BEGIN_NODE is equivalent to IPM_BEGIN_HOST in the node program. It performs all the initializations needed for any other IPM call.
See also IPM_INIT.

Example

```
      program node
c
include(ipm.h)
c
      IPM_DECLAR
c
      IPM_BEGIN_NODE
c
      write(6,*) 'All these values are available:',
     *           IPM_RANK, IPM_HOST, IPM_DADDY, IPM_ROOT,
     *           IPM_TOPO, IPM_NB_SONS, IPM_NB_NEIGHBORS,
     *           'and so on...'
c
      IPM_END
      end
```

Synopsis

IPM_CREATE_GRAPH(name_exe,nb_nodes,index,neighbors,host_status,numc)

Arguments

IN

name_exe	-	character string that contains the name of the executable to be spawned.
nb_nodes	-	number of nodes in the graph.
index	-	vector of integers containing indices in neighbors.
neighbors	-	vector of integers containing neighbors for all graph nodes.
host_status	-	integer which specifies if the host is included in the graph or not.

OUT

numc	-	integer equal to the number of created nodes.

Discussion

The mapping function IPM_CREATE_GRAPH allows the user to define a customized graph for its processes. *index* starts from 0 if *host_status* is set to 1, and starts at 1 otherwise. The last value of *index* must point to the position just after the last element of *neighbors*. Neighbors of node number k are *neighbors(index(k))* up to *neighbors(index(k+1)-1)*.
See also IPM_HOST_STATUS, IPM_NB_NEIGHBORS, IPM_NEIGHBORS.

Example

```
c*    host + a 3 nodes graph
      integer index(4), neighbors(4)
c
      index(1)=1
      neighbors(1)=2
      neighbors(2)=3
      index(2)=3
      neighbors(3)=1
      index(3)=4
      neighbors(4)=1
      index(4)=5
      IPM_CREATE_GRAPH('node',3,index,neighbors,IPM_HOST_0,numc)
```

Synopsis

IPM_CREATE_TORUS(name_exe,dim,ndims,host_status,numc)

Arguments

IN

name_exe	-	character string that contains the name of the executable to be spawned.
dim	-	integer which specifies the dimension of the torus.
ndims	-	vector of integers containing the size of the torus in each direction.
host_status	-	integer which specifies if the host is included in the torus or not.

OUT

numc	-	integer equal to the number of created nodes.

Discussion

The mapping function IPM_CREATE_TORUS defines a cartesian topology where each process is associated to *dim* coordinates. If *host_status* is equal to 1, the host process is included in the torus topology and its coordinates are (0,..,0). In this case the number of nodes created *numc* is equal to the product of the *ndims* entries minus 1. The vector *ndims* should contain at least *dim* elements, with *dim* inferior or equal to 4.
See also IPM_HOST_STATUS, IPM_TOPO,
IPM_TORDIM, IPM_TORSIZE, IPM_GET_COORDS.

Example

```
c*    (3,2) torus
      integer dim, ndims(2)
c
      dim = 2
      ndims(1) = 3
      ndims(2) = 2
      IPM_CREATE('node',dim,ndims,IPM_HOST_I,numc)
c     ...numc should be equal to 5
```

Synopsis

IPM_DADDY

Arguments

IN

none

OUT

none

Discussion

IPM_DADDY is the rank process of the logical father of the calling process in the global binary tree. Host and root processes don't have any father process, thus IPM_DADDY is set to IPM_NO_DADDY.
See also, IPM_ROOT, IPM_NB_SONS, IPM_SONS.

Example

```
c
      if ( IPM_DADDY.eq.IPM_NO_DADDY ) then
         write(6,*) 'I am probably the host process...'
         if ( IPM_RANK.eq.IPM_ROOT ) then
            write(6,*) 'bouuhh, I am only root!'
         else
            write(6,*) 'yeaahh, I am the master!'
         endif
      endif
c
```

Synopsis

IPM_DECLAR

Arguments

IN

none

OUT

none

Discussion

IPM_DECLAR should be included in user's declarations in each routine which invokes IPM functions or subroutines. It contains internal declarations.

Example

```
      program short
c
include(ipm.h)
c
c* the shortest IPM program
c
      IPM_DECLAR
      IPM_INIT
      write(6,*) 'I am the process', IPM_RANK
      IPM_END
      end
```

Synopsis

IPM_END

Arguments

IN

none

OUT

none

Discussion

IPM_END exits from the parallel application but doesn't kill the process.

Example

```
      program TheEnd
c
include(ipm.h)
c
      IPM_DECLAR
      IPM_END
      end
```

Synopsis

IPM_GBARRIER(msgtag)

Arguments

IN

bartag - integer barrier identifier supplied by the user. Should be in the same range than message tag. It allows the user to distinguish between two barriers.

OUT

none

Discussion

The IPM_GBARRIER call synchronizes all the processes, including the host process. It is recommended to do not use the same tag number in two consecutive barriers.
See also IPM_BARRIER.

Example

```
c
      integer bartag
c
      bartag = 10
      IPM_BARRIER(bartag)
      IPM_GBARRIER(bartag+10)
c
```

Synopsis

IPM_GET_ALL_COORDS(coords)

Arguments

IN

none

OUT

coords - vector of integers containing coordinates of all the active processes.

Discussion

IPM_GET_ALL_COORDS returns a vector of integers filled with the coordinates of all the processes ordered by their rank number in a torus topology. If host is active in the topology, *coords* contains IPM_SIZE * IPM_TORDIM elements, IPM_NODE * IPM_TORDIM otherwise.
See also IPM_SIZE, IPM_TORDIM, IPM_GET_COORDS.

Example

```
      integer xy(2), coords(12)
c
c*    in a (3,2) torus...
c
      IPM_GET_ALL_COORDS(coords)
      IPM_GET_COORDS(6,xy)
      if ( xy(1).ne.coords(11).or.xy(2).ne.coords(12) ) then
         write(6,*) 'error in IPM coords functions'
         abort
      end
```

Synopsis

IPM_GET_COORDS(rank,coords)

Arguments

IN

rank - logical rank of the process

OUT

coords - vector of integers containing the coordinates of the rank-th process.

Discussion

Coordinates in direction k start from 0 up to IPM_TORSIZE(k)-1. The last dimension varies first. In a (2,2) grid, for example, we have:
process 0 - (0,0)
process 1 - (0,1)
process 2 - (1,0)
process 3 - (1,1)
See also IPM_GET_ALL_COORDS, IPM_GET_RANK.

Example

```
      integer coords(2)
c
      IPM_GET_COORDS(1,coords)
      if ( coords(1)+coords(2).ne.0 ) then
         write(6,*) 'host is active in the torus'
      endif
```

Synopsis

IPM_GET_RANK(coords,rank)

Arguments

IN

coords - vector of integers containing coordinates of a process.

OUT

rank - logical rank of the process associated to the specified coordinates.

Discussion

IPM_GET_RANK translates the process coordinates to process ranks as they are used in communications routines.

Example

```
      integer coords(2), rank
c
      IPM_GET_COORDS(1,coords)
      IPM_GET_RANK(coords,rank)
c
      if ( rank.ne.1 ) then
         write(6,*) 'error in IPM COORDS<-->RANK routines'
         abort
      endif
```

Synopsis

IPM_HOST

Arguments

IN

none

OUT

none

Discussion

IPM_HOST returns the process rank of the host process. Even if this value is often set to 0, the user should always use the IPM_HOST function in communication routines.

Example

```
c
c*    receive initial value from host
c
      integer initv, is, it, il
c
      IPM_RECV(IPM_HOST,IPM_ANYTAG,IPM_INTEGER,1,initv,is,it,il)
```

Synopsis

IPM_HOST_STATUS

Arguments

IN

none

OUT

none

Discussion

The IPM_HOST_STATUS returns the current status of the host process in the virtual topology (as defined by the IPM_CREATE routines):

IPM_HOST_I: *host active*
IPM_HOST_O: *host passive*

See also IPM_CREATE_TORUS, IPM_CREATE_GRAPH.

Example

```
c
c*    some social considerations...
c
      if ( IPM_HOST_STATUS.eq.IPM_HOST_O ) then
         write(6,*) 'master is sleeping while the slaves are working'
      endif
c
```

Synopsis

IPM_INIT

Arguments

IN

none

OUT

none

Discussion

IPM_INIT is a first attempt to MPI-like interface. Its allows full SPMD programming model by automatically checking if the call is done from the host or node program and then starting IPM_BEGIN_HOST or IPM_BEGIN_NODE.

Example

```
      program spmd
c
include(ipm.h)
c
      IPM_DECLAR
      IPM_INIT
      if ( IPM_RANK.eq.IPM_HOST ) then
         write(6,*) 'It is great to wake up and to discover that'
         write(6,*) 'you are the master!'
      endif
c
      IPM_END
      end
```

Synopsis

IPM_LIBRARY

Arguments

IN

none

OUT

none

Discussion

IPM_LIBRARY returns the running parallel library, defined by one of the following integer values:

IPM_PARMACS
IPM_PVM

Example

```
c*    parvm or p-macs?
c
      if (IPM_LIBRARY.eq.IPM_PARMACS) then
         write(6,*) 'the FunParTools advice: try PVM !'
      else
         write(6,*) 'the FunParTools advice: try PARMACS !'
      endif
```

Synopsis

IPM_MCAST(nb_dests,dests,msgtag,type,lenmsg,buffer)

Arguments

IN

nb_dests	-	number of recipients of the message (integer).
dests	-	vector of integers containing the ranks of recipients' processes
msgtag	-	integer message tag identifier.
type	-	data type (integer).
lenmsg	-	lengh in elements of the message (integer).
buffer	-	buffer containing the message.

OUT

none

Discussion

The multicasting call allows the user to send the same message to various processes. See also IPM_SEND for the list of datatypes.

Example

```
      integer dests(3), buff
c
      dests(1) = 4
      dests(2) = IPM_HOST
      dests(3) = 7
      IPM_MCAST(3,dests,1,IPM_INTEGER,1,buff)
```

Synopsis

IPM_MSG_RECV

Arguments

IN

none

OUT

none

Discussion

IPM_MSG_RECV returns the number of messages already received by the calling process.

Example

```
c*    to be or not to be a parallel program...
c
      if (IPM_MSG_RECV.eq.0) then
         write(6,*) 'Hey! are you sure to run a parallel program?'
      endif
```

Synopsis

IPM_MSG_SEND

Arguments

IN

none

OUT

none

Discussion

IPM_MSG_SEND returns the number of messages already sent by the calling process.

Example

```
c*    looking for White-snow
c
      if (IPM_MSG_SEND.eq.0) then
         write(6,*) 'i am the grumpy dwarf!'
      endif
```

Synopsis

IPM_NBRECV

Arguments

IN

none

OUT

none

Discussion

IPM_NBRECV returns the number of bytes already received by the calling process.

Example

```
c*    the FunParTools process...
c
      if (IPM_NBRECV.eq.0) then
         write(6,*) 'Good! thanks God there is nothing to do yet!'
      endif
```

Synopsis

IPM_NBSEND

Arguments

IN

none

OUT

none

Discussion

IPM_NBSEND returns the number of bytes already received by the calling process.

Example

```
c*    a good load balance...
c
      if (IPM_NBRECV.gt.IPM_NBSEND) then
         IPM_SEND(IPM_HOST,1,IPM_BYTE,IPM_NBRECV-IPM_NBSEND,ibuff)
         if (IPM_NBRECV.ne.IPM_NBSEND) then
            write(6,*) 'Another bug in IPM !!!'
         endif
      endif
```

Synopsis

IPM_NB_NEIGHBORS

Arguments

IN

none

OUT

none

Discussion

If a graph topology has been defined, IPM_NB_NEIGHBORS returns the number of neighbors of the calling process. If there is no graph defined, this value is set to 0.
See also IPM_CREATE_GRAPH, IPM_TOPO, IPM_NEIGHBORS.

Example

```
c
c*    suspicious process...
c
      write(6,*) 'I have ', IPM_NB_NEIGHBORS, ' guys hanging around'
```

Synopsis

IPM_NB_SONS

Arguments

IN

none

OUT

none

Discussion

The IPM_NB_SONS function returns the number of sons of the calling process in the global tree. This value is less than or equal to IPM_MAXSONS.
See also IPM_ROOT, IPM_DADDY, IPM_SONS.

Example

```
c
c*    Sweet family...
c
      if ( IPM_NB_SONS.eq.0.and.IPM_DADDY.eq.IPM_NO_DADDY ) then
         write(6,*) 'Hey! is there someone here?'
         write(6,*) 'I am the lonesome process...'
      endif
```

Synopsis

IPM_NEIGHBORS(num)

Arguments

IN

num - num-th neighbor of the calling process (integer).

OUT

none

Discussion

The IPM_NEIGHBORS function returns the *num*-th neighbor of the calling process in the logical graph topology. *num* varies from 1 up to IPM_NB_NEIGHBORS. See also, IPM_CREATE_GRAPH, IPM_NB_NEIGHBORS.

Example

```
      integer  htime, invitation
c
c*    the party will start at 18h30
      invitation = 1
      htime      = 1830
c
      do 100 i=1,IPM_NB_NEIGHBORS
         IPM_SEND(IPM_NEIGHBORS(i),invitation,IPM_INTEGER,1,htime)
  100 continue
```

Synopsis

IPM_NODES

Arguments

IN

none

OUT

none

Discussion

IPM_NODES returns the number of nodes that have been created by one of the IPM_CREATE functions, 0 otherwise.
See also, IPM_SIZE.

Example

```
if ( IPM_NODES.eq.0 ) then
   write(6,*) 'it is not St Tropez here!'
endif
```

Synopsis

IPM_NRECV(reqrank,reqtag,type,len,buffer,recvrank,recvtag,recvlen,arrived)

Arguments

IN

reqrank	-	requested rank of the sending process (integer).
reqtag	-	requested tag of the expected message (integer).
type	-	data type expected (integer).
len	-	number of elements expected (integer).
buffer	-	starting address of the receive buffer (handle).

OUT

recvrank	-	rank of the effective sending process (integer).
recvtag	-	tag of the received message (integer).
recvlen	-	lengh in bytes of the received message (integer).
arrived	-	integer which specifies if the requested message has arrived (1) or not (0).

Discussion

IPM_NRECV is used for asynchronous message passing between any two active processes. It performs a non-blocking receive: the calling process checks if a message with reqtag tag from reqrank is arrived or not and returns immediately. Wildcards IPM_ANYSND and IPM_ANYTAG can be used in place of the first two arguments, matching for any sender or any tag.
See also IPM_SEND for the list of datatypes, IPM_ANYSND, IPM_ANYTAG, IPM_RECV, IPM_TYPE_LENGTH.

Example

```
c*    looking for a job
c
 100  continue
      IPM_NRECV(IPM_ANYSND,1,IPM_INTEGER,1,ijob,is,it,il,iok)
      if (iok.eq.0)
c        no job available
         ip = sleep(60)
         go to 100
      else
c        have a big nap before working
         ip = sleep (7200)
      endif
```

Synopsis

IPM_PROBE(reqrank,reqtag,recvrank,recvtag,arrived)

Arguments

IN

reqrank - requested rank of the sending process (integer).
reqtag - requested tag of the expected message (integer).

OUT

recvrank - rank of the effective sending process, if any (integer).
recvtag - tag of the received message, if any (integer).
arrived - integer which specifies if the requested message has arrived (1) or not (0).

Discussion

IPM_PROBE checks if a message from the *reqrank* process with a *reqtag* tag identifier has arrived or not. If such a message has arrived *recvrank* and *recvtag* return the rank of the sending process and the tag of the message. This allows use of wildcards IPM_ANYSND and IPM_ANYTAG for the first arguments.
See also, IPM_ANYSND, IPM_ANYTAG.

Example

```
c
c*    Probe if I have received any letter from daddy
c
      IPM_PROBE(IPM_DADDY,IPM_ANYTAG,is,it,iok)
      if ( iok.eq.0 ) then
         write(6,*) 'Daddy has probably lost my address!'
      endif
```

Synopsis

IPM_RANK

Arguments

IN

none

OUT

none

Discussion

IPM_RANK returns the logical identifier of the calling process. Processes are numbered from 0 up to IPM_SIZE-1. These values are used is communications and probe functions.

Example

```
write(6,*) 'my rank=', IPM_RANK
```

Synopsis

IPM_RECV(reqrank,reqtag,type,len,buffer,recvrank,recvtag,recvlen)

Arguments

IN

reqrank	-	requested rank of the sending process (integer).
reqtag	-	requested tag of the expected message (integer).
type	-	data type expected (integer).
len	-	number of elements expected (integer).
buffer	-	starting address of the receive buffer (handle).

OUT

recvrank	-	rank of the effective sending process (integer).
recvtag	-	tag of the received message (integer).
recvlen	-	lengh in bytes of the received message (integer).

Discussion

IPM_RECV is used for asynchronous message passing between any two active processes. The calling process is blocked until an appropriate message has been received. Wildcards IPM_ANYSND and IPM_ANYTAG can be used in place of the first two arguments, matching for any sender or any tag.
See also IPM_SEND for the list of datatypes, IPM_ANYSND, IPM_ANYTAG, IPM_TYPE_LENGTH.

Example

```
c*    receive from host basic dimensions of Kim Basinger.
      integer dim(3)
c
      IPM_RECV(IPM_HOST,IPM_ANYTAG,IPM_INTEGER,3,dim,is,it,il)
      if ( dim(1).lt.90.or.dim(2).gt.68.or.dim(3).lt.85 ) then
         write(6,*) 'I received incoherent initial values'
         write(6,*) 'for the modelisation of Kim Basinger's body.'
         IPM_ABORT
      endif
```

Synopsis

IPM_ROOT

Arguments

IN

none

OUT

none

Discussion

IPM_ROOT returns the rank of the root process in the global tree provided by IPM for global operations.
See also, IPM_DADDY, IPM_NB_SONS, IPM_SONS.

Example

```
write(6,*) 'root=', IPM_ROOT
```

Synopsis

IPM_SEND(rank,msgtag,type,len,buffer)

Arguments

IN

rank	-	rank identifier of destination (integer).
msgtag	-	message tag identifier (integer).
type	-	date type (integer).
len	-	lengh of the message in elements (integer).
buffer	-	starting address of the message (handle).

OUT

none

Discussion

IPM_SEND is used for asynchronous message-passing between any two active processes. The calling process continues as soon as the message is safely on its way.
One could use generic datatypes such as:

IPM_CHARACTER
IPM_BYTE
IPM_INTEGER
IPM_REAL
IPM_DBLE
IPM_COMPLEX

or specific datatypes:

IPM_INT2, IPM_INT4, IPM_INT8
IPM_REAL4, IPM_REAL8
IPM_COMPLEX8, IPM_COMPLEX16

See also, IPM_RECV, IPM_TYPE_LENGTH.

Example

```
c*    nothing to send...
      integer null
      null = 0
c
      IPM_SEND(IPM_HOST,1,IPM_INTEGER,1,null)
```

Synopsis

IPM_SIZE

Arguments

IN

none

OUT

none

Discussion

IPM_SIZE returns the total number of processes in the application, including host process. Users should be aware that this value is not dynamically updated if a process leaves the application.
See also IPM_NODES.

Example

```
if ( IPM_SIZE.eq.1 ) then
   write(6,*) 'application not yet started'
endif
```

Synopsis

IPM_SONS(num)

Arguments

IN

num - num-th son of the calling process (integer).

OUT

none

Discussion

The IPM_SONS function returns the rank of the *num*-th son of the calling process in the global tree. *num* varies from 1 up to IPM_NB_SONS.
See also IPM_ROOT, IPM_DADDY, IPM_NB_SONS.

Example

```
c*    first day of the month, it's time for allowance.
c
      integer i, allowance
      allowance = 1000 / IPM_NB_SONS
c
      do 100 i=1,IPM_NB_SONS
         IPM_SEND(IPM_SONS(i),1,IPM_INT4,1,allowance)
  100 continue
```

Synopsis

IPM_TOPO

Arguments

IN

none

OUT

none

Discussion

IPM_TOPO returns the active topology, which can be one the following predefined integer values:

IPM_NONE
IPM_TORUS
IPM_GRAPH

Example

```
if ( IPM_TOPO.eq.IPM_NONE ) then
   write(6,*) 'no mapping defined'
endif
```

Synopsis

IPM_TORDIM

Arguments

IN

none

OUT

none

Discussion

IPM_TORDIM returns the torus dimension. Even if the active topology is not a torus, a logical one-dimensional torus is always provided and all IPM cartesian functions are allowed.
See also, IPM_CREATE_TORUS, IPM_TORSIZE.

Example

```
write(6,*) 'dimension of the torus=', IPM_TORDIM
```

Synopsis

IPM_TORSIZE(num)

Arguments

IN

num - integer which specifies a direction of the torus.

OUT

none

Discussion

The IPM_TORSIZE function returns the dimension of the torus in *num*-th direction.
See also IPM_CREATE_TORUS, IPM_TORDIM.

Example

```
      do 100 i=1,IPM_TORDIM
         write(6,*) 'Size in direction ',i,' = ',IPM_TORSIZE(i)
  100 continue
```

Synopsis

IPM_TYPE_LENGTH(datatype)

Arguments

IN

datatype - specified datatype.

OUT

none

Discussion

IPM_TYPE_LENGTH returns the length in bytes of the specified datatype. Generic dataypes can be setted by explicit changes in the ipm.h file before compilation. By default, IPM_INTEGER is IPM_INT4, IPM_REAL is IPM_REAL4, and IPM_COMPLEX is IPM_COMPLEX8.

Example

```
c
      if ( IPM_TYPE_LENGTH(IPM_REAL).eq.8 ) then
         write(6,*) 'Double precision is implicit!'
      endif
c
```

Synopsis

IPM_VERSION

Arguments

IN

none

OUT

none

Discussion

IPM_VERSION returns a real number which specifies the current version of the IPM package.

Example

```
c*    some wishes...
c
      write(6,*) 'Is it possible to have a ',IPM_VERSION+0.1,
     *           'IPM version without so many bugs?!'
```

Appendix II:
An example of generic makefile using IPM

```
#
# Makefile for programs under IPM
# (c) Eric Sevault, Vinvent Van Kemenade
#     CERFACS
#
# This makefile should be Ok for these computers:
#
#       RS6K : IBM RS6000
#       SUN4 : Sparc workstation
#       CRAY : C90, CRAY-2
#       MCS2 : MEIKO CS-2 (have fun!)
#
# With these message-passing librairies:
#
#       PVM  : PVM 3.2.x or 3.3.x              make pvm
#       PCS  : PVM Meiko CS-2                  make pvmcs
#       PM6  : PARMACS 6.0                     make pm6
#
# ipm location :
#
IPM       = $(HOME)/ipm/include
#
# C Compiler if needed :
#
CC_RS6K =       c89
CC_SUN4 =       gcc
CC_CRAY =       cc
CC_MCS2 =       cc
CC      =       $(CC_$(PVM_ARCH))
#
# Fortan Compiler :
#
FF_RS6K =       xlf
FF_SUN4 =       /usr/lang/f77
FF_CRAY =       cf77
FF_MCS2 =       f77
F77     =       $(FF_$(PVM_ARCH))
#
# Flags when compiling :
#
FL_RS6K =       -O
FL_SUN4 =       -O
FL_CRAY =       -O1
FL_MCS2 =       -g
FFLAGS  =       $(FL_$(PVM_ARCH))
#
# Flags when linking :
```

```
#
LF_RS6K =       -LSP
LF_SUN4 =
LF_CRAY =
LF_MCS2 =       -g
LFLAGS  =       $(LF_$(PVM_ARCH))
#
# m4 location :
#
M4_RS6K =       /usr/bin/m4
M4_SUN4 =       /usr/bin/m4
M4_CRAY =       /usr/bin/m4
M4_MCS2 =       /usr/ccs/bin/m4
M4DIR   =       $(M4_$(PVM_ARCH))
M4FILTER=       egrep '[0-9]|[a-z]|[A-Z]'
#
# Message passing librairies
# --------------------------
#
# PARMACS 6.0
#
# 1 - Meiko CS-2
PP_MCS2 =       meiko
PR_MCS2 =       cs2
PD_MCS2 =       /opt/MEIKOcs2/parmacs
#
# 2 - Cluster IBM's
PP_RS6K =       ws
PR_RS6K =       RS6K
PD_RS6K =       /usr/parmacs

PARALLEL_PLATFORM = $(PP_$(PVM_ARCH))
PARALLEL_RESOURCE = $(PR_$(PVM_ARCH))
PARMACS_DIR       = $(PD_$(PVM_ARCH))

INCLUDE_DIR  = $(PARMACS_DIR)/include/$(PARALLEL_PLATFORM)
LIB_DIR      = $(PARMACS_DIR)/lib/$(PARALLEL_PLATFORM)/$(PARALLEL_RESOURCE)
PM6_SYS1     = -L/opt/MEIKOcs2/lib
PM6_SYS2     = -lrms -lsocket -lnsl -lew -lelan

SYS_MCS2=       $(PM6_SYS1) $(PM6_SYS2)
SYS_RS6K=       -lrpcsvc

PM6LIBC =       $(LIB_DIR)/libpm6.a
PM6LIBF =
PM6     =       $(PM6LIBF) $(PM6LIBC) $(SYS_$(PVM_ARCH))
RUNPM6  =       $(HOME)/ipm/example
```

```
#
# PVM standart
#
PVMLIBC =       $(PVM_ROOT)/lib/$(PVM_ARCH)/libpvm3.a
PVMLIBF =       $(PVM_ROOT)/lib/$(PVM_ARCH)/libfpvm3.a
PVM     =       $(PVMLIBF) $(PVMLIBC)
RUNPVM  =       $(HOME)/pvm3/bin/$(PVM_ARCH)
#
# PVM &#%$&@ CS-2
#
LOCAL_ROOT=       /opt/MEIKOcs2
PCSLIBC   =       $(LOCAL_ROOT)/lib/libpvm3.a
PCSLIBF   =       $(LOCAL_ROOT)/lib/libfpvm3.a
PCSLIBA   =       $(PCSLIBF) $(PCSLIBC) -lsocket -lnsl
LDCSFL    =       -L$(LOCAL_ROOT)/lib -lrms -lew -lelan
PCS       =       $(PCSLIBA) $(LDCSFL)
RUNPCS    =       $(HOME)
#
# COMPILATION
# -----------
#
CMD       =       main

OBJS      =       obj1.0 obj2.o

.SUFFIXES: .o .f .F

all:        ipm.h expand $(CMD)
#
ipm.h:
       echo 'You must select a message-passing style'
expand:
       mkdir expand
#
#  - PVM -
#
pvm:   OK.pvm all
       $(F77) $(LFLAGS) -o $(CMD) $(OBJS) $(PVM)
       mv $(CMD) $(RUNPVM)
OK.pvm:
       ln -s $(IPM)/ipm.h.pvm ipm.h
       ln -s $(PVM_ROOT)/include/fpvm3.h fpvm3.h
       touch OK.pvm
#
#  - PVM CS-2 -
#
pvmcs: OK.pvmcs all
```

```
	$(F77) $(LFLAGS) -o $(CMD) $(OBJS) $(PCS)
OK.pvmcs:
	ln -s $(IPM)/ipm.h.pvm ipm.h
	ln -s /opt/MEIKOcs2/include/PVM/fpvm3.h fpvm3.h
	touch OK.pvmcs
#
#  - PARMACS 6.0 -
#
pm6:   OK.pm6 all
	$(F77) $(LFLAGS) -o $(CMD) $(OBJS) $(PM6)
OK.pm6:
	ln -s $(INCLUDE_DIR)/pm.inc pm.inc
	ln -s $(IPM)/ipm.h.pm6 ipm.h
	touch OK.pm6
#
#  - Compilations -
#
$(CMD):	$(OBJS)
#
.f.o :
	$(F77) $(FFLAGS) -c $*.f
#
.F.o:
	$(M4DIR) $*.F | egrep -v '#' | $(M4FILTER) > $*.f
	$(F77) $(FFLAGS) -c $*.f
	mv $*.f expand
#
.c.o :
	$(CC)  $(CFLAGS) -c $*.c
#
clean:
	rm -f $(CMD)
	rm -f *.o
	rm -f ipm.h fpvm3.h pm.inc
	rm -f OK.*
	rm -r expand
```